2 *Supervised Learning*

We discuss supervised learning starting from the simplest case, which is learning a class from its positive and negative examples. We generalize and discuss the case of multiple classes, then regression, where the outputs are continuous.

2.1 Learning a Class from Examples

LET US say we want to learn the *class*, C, of a "family car." We have a set of examples of cars, and we have a group of people that we survey to whom we show these cars. The people look at the cars and label them; the cars that they believe are family cars are *positive examples*, and the other cars are *negative examples*. Class learning is finding a description that is shared by all positive examples and none of the negative examples. Doing this, we can make a prediction: Given a car that we have not seen before, by checking with the description learned, we will be able to say whether it is a family car or not. Or we can do knowledge extraction: This study may be sponsored by a car company, and the aim may be to understand what people expect from a family car.

POSITIVE EXAMPLES
NEGATIVE EXAMPLES

After some discussions with experts in the field, let us say that we reach the conclusion that among all features a car may have, the features that separate a family car from other cars are the price and engine power. These two attributes are the *inputs* to the class recognizer. Note that when we decide on this particular *input representation*, we are ignoring various other attributes as irrelevant. Though one may think of other attributes such as seating capacity and color that might be important for distinguishing among car types, we will consider only price and engine power to keep this example simple.

INPUT REPRESENTATION

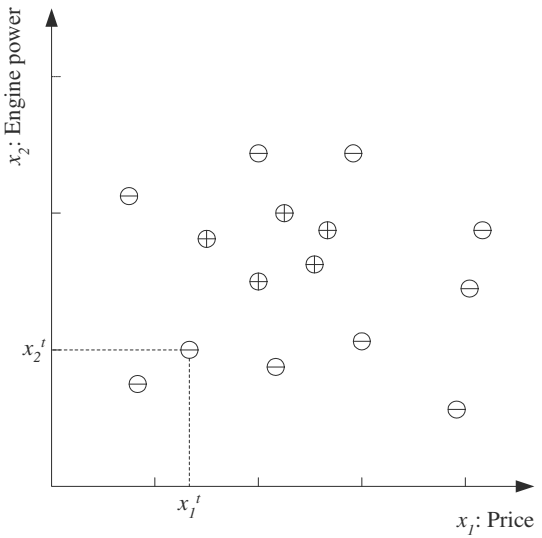

Figure 2.1 Training set for the class of a "family car." Each data point corresponds to one example car, and the coordinates of the point indicate the price and engine power of that car. '+' denotes a positive example of the class (a family car), and '−' denotes a negative example (not a family car); it is another type of car.

Let us denote price as the first input attribute x_1 (e.g., in U.S. dollars) and engine power as the second attribute x_2 (e.g., engine volume in cubic centimeters). Thus we represent each car using two numeric values

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{2.1}$$

and its label denotes its type

$$r = \begin{cases} 1 & \text{if } \mathbf{x} \text{ is a positive example} \\ 0 & \text{if } \mathbf{x} \text{ is a negative example} \end{cases} \tag{2.2}$$

Each car is represented by such an ordered pair $(\mathbf{x}, r)$ and the training set contains N such examples

$$\mathcal{X} = \{\mathbf{x}^t, r^t\}_{t=1}^N \tag{2.3}$$

where t indexes different examples in the set; it does not represent time or any such order.

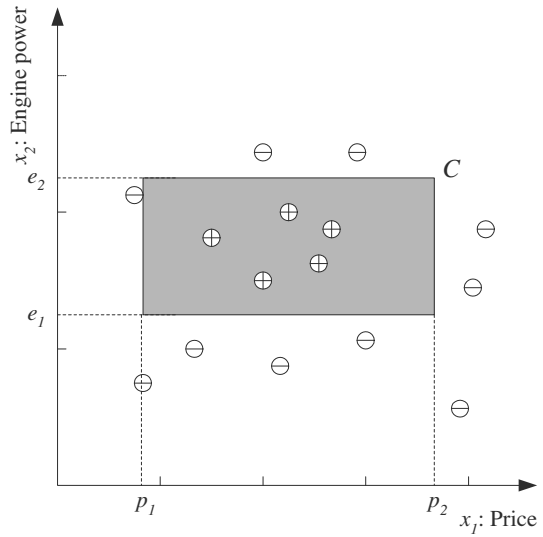

Figure 2.2 Example of a hypothesis class. The class of family car is a rectangle in the price-engine power space.

Our training data can now be plotted in the two-dimensional (x_1, x_2) space where each instance t is a data point at coordinates (x_1^t, x_2^t) and its type, namely, positive versus negative, is given by r^t (see figure 2.1).

After further discussions with the expert and the analysis of the data, we may have reason to believe that for a car to be a family car, its price and engine power should be in a certain range

$$(p_1 \leq \text{price} \leq p_2) \text{ AND } (e_1 \leq \text{engine power} \leq e_2) \tag{2.4}$$

for suitable values of p_1, p_2, e_1, and e_2. Equation 2.4 thus assumes C to be a rectangle in the price-engine power space (see figure 2.2).

HYPOTHESIS CLASS

Equation 2.4 fixes $\mathcal{H}$, the *hypothesis class* from which we believe C is drawn, namely, the set of rectangles. The learning algorithm then finds the particular *hypothesis*, $h \in \mathcal{H}$, to approximate C as closely as possible.

HYPOTHESIS

Though the expert defines this hypothesis class, the values of the parameters are not known; that is, though we choose $\mathcal{H}$, we do not know which particular $h \in \mathcal{H}$ is equal, or closest, to C. But once we restrict our

attention to this hypothesis class, learning the class reduces to the easier problem of finding the four parameters that define h.

The aim is to find $h \in \mathcal{H}$ that is as similar as possible to C. Let us say the hypothesis h makes a prediction for an instance $\boldsymbol{x}$ such that

$$h(\boldsymbol{x}) = \begin{cases} 1 & \text{if } h \text{ classifies } \boldsymbol{x} \text{ as a positive example} \\ 0 & \text{if } h \text{ classifies } \boldsymbol{x} \text{ as a negative example} \end{cases} \tag{2.5}$$

In real life we do not know $C(\boldsymbol{x})$, so we cannot evaluate how well $h(\boldsymbol{x})$ matches $C(\boldsymbol{x})$. What we have is the training set $\mathcal{X}$, which is a small subset of the set of all possible $\boldsymbol{x}$. The *empirical error* is the proportion of training instances where *predictions* of h do not match the *required values* given in $\mathcal{X}$. The error of hypothesis h given the training set $\mathcal{X}$ is

EMPIRICAL ERROR

$$E(h|\mathcal{X}) = \sum_{t=1}^{N} 1(h(\boldsymbol{x}^t) \neq r^t) \tag{2.6}$$

where $1(a \neq b)$ is 1 if $a \neq b$ and is 0 if $a = b$ (see figure 2.3).

In our example, the hypothesis class $\mathcal{H}$ is the set of all possible rectangles. Each quadruple $(p_1^h, p_2^h, e_1^h, e_2^h)$ defines one hypothesis, h, from $\mathcal{H}$, and we need to choose the best one, or in other words, we need to find the values of these four parameters given the training set, to include all the positive examples and none of the negative examples. Note that if x_1 and x_2 are real-valued, there are infinitely many such h for which this is satisfied, namely, for which the error, E, is 0, but given a future example somewhere close to the boundary between positive and negative examples, different candidate hypotheses may make different predictions. This is the problem of *generalization*—that is, how well our hypothesis will correctly classify future examples that are not part of the training set.

GENERALIZATION

MOST SPECIFIC HYPOTHESIS

One possibility is to find the *most specific hypothesis*, S, that is the tightest rectangle that includes all the positive examples and none of the negative examples (see figure 2.4). This gives us one hypothesis, $h = S$, as our induced class. Note that the actual class C may be larger than S but is never smaller. The *most general hypothesis*, G, is the largest rectangle we can draw that includes all the positive examples and none of the negative examples (figure 2.4). Any $h \in \mathcal{H}$ between S and G is a valid hypothesis with no error, said to be *consistent* with the training set, and such h make up the *version space*. Given another training set, S, G, version space, the parameters and thus the learned hypothesis, h, can be different.

MOST GENERAL HYPOTHESIS

VERSION SPACE

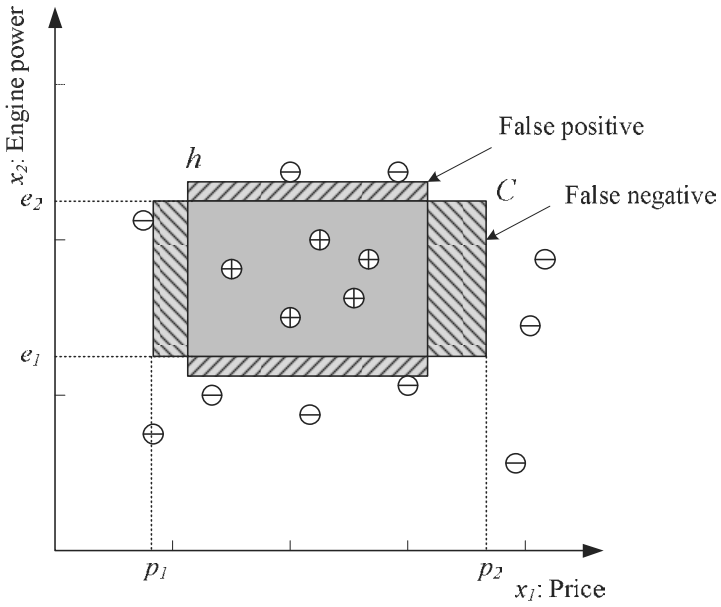

Figure 2.3 C is the actual class and h is our induced hypothesis. The point where C is 1 but h is 0 is a false negative, and the point where C is 0 but h is 1 is a false positive. Other points—namely, true positives and true negatives—are correctly classified.

Actually, depending on $\mathcal{X}$ and $\mathcal{H}$, there may be several S_i and G_j which respectively make up the S-set and the G-set. Every member of the S-set is consistent with all the instances, and there are no consistent hypotheses that are more specific. Similarly, every member of the G-set is consistent with all the instances, and there are no consistent hypotheses that are more general. These two make up the boundary sets and any hypothesis between them is consistent and is part of the version space. There is an algorithm called candidate elimination that incrementally updates the S- and G-sets as it sees training instances one by one; see Mitchell 1997. We assume $\mathcal{X}$ is large enough that there is a unique S and G.

Given $\mathcal{X}$, we can find S, or G, or any h from the version space and use it as our hypothesis, h. It seems intuitive to choose h halfway between S and G; this is to increase the *margin*, which is the distance between the

MARGIN

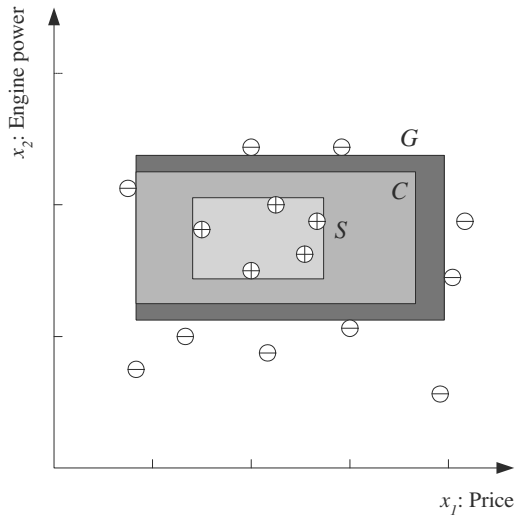

Figure 2.4 S is the most specific and G is the most general hypothesis.

boundary and the instances closest to it (see figure 2.5). For our error function to have a minimum at h with the maximum margin, we should use an error (loss) function which not only checks whether an instance is on the correct side of the boundary but also how far away it is. That is, instead of $h(\boldsymbol{x})$ that returns 0/1, we need to have a hypothesis that returns a value which carries a measure of the distance to the boundary and we need to have a loss function which uses it, different from $1(\cdot)$ that checks for equality.

In some applications, a wrong decision may be very costly and in such a case, we can say that any instance that falls in between S and G is a case of *doubt*, which we cannot label with certainty due to lack of data. In such a case, the system *rejects* the instance and defers the decision to a human expert.

DOUBT

Here, we assume that $\mathcal{H}$ includes C; that is, there exists $h \in \mathcal{H}$, such that $E(h|\mathcal{X})$ is 0. Given a hypothesis class $\mathcal{H}$, it may be the case that we cannot learn C; that is, there exists no $h \in \mathcal{H}$ for which the error is 0. Thus, in any application, we need to make sure that $\mathcal{H}$ is flexible enough, or has enough "capacity," to learn C.

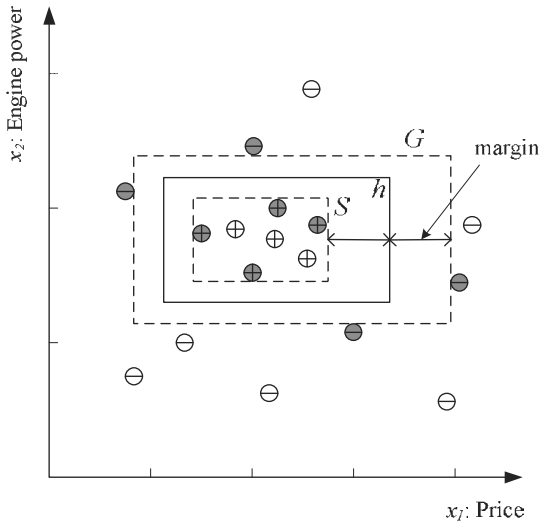

Figure 2.5 We choose the hypothesis with the largest margin, for best separation. The shaded instances are those that define (or support) the margin; other instances can be removed without affecting h.

2.2 Vapnik-Chervonenkis (VC) Dimension

Let us say we have a dataset containing N points. These N points can be labeled in 2^N ways as positive and negative. Therefore, 2^N different learning problems can be defined by N data points. If for any of these problems, we can find a hypothesis $h \in \mathcal{H}$ that separates the positive examples from the negative, then we say $\mathcal{H}$ *shatters* N points. That is, any learning problem definable by N examples can be learned with no error by a hypothesis drawn from $\mathcal{H}$. The maximum number of points that can be shattered by $\mathcal{H}$ is called the *Vapnik-Chervonenkis* (VC) *dimension* of $\mathcal{H}$, is denoted as $VC(\mathcal{H})$, and measures the *capacity* of $\mathcal{H}$.

VC DIMENSION

In figure 2.6, we see that an axis-aligned rectangle can shatter four points in two dimensions. Then $VC(\mathcal{H})$, when $\mathcal{H}$ is the hypothesis class of axis-aligned rectangles in two dimensions, is four. In calculating the VC dimension, it is enough that we find four points that can be shattered; it is not necessary that we be able to shatter *any* four points in two di-

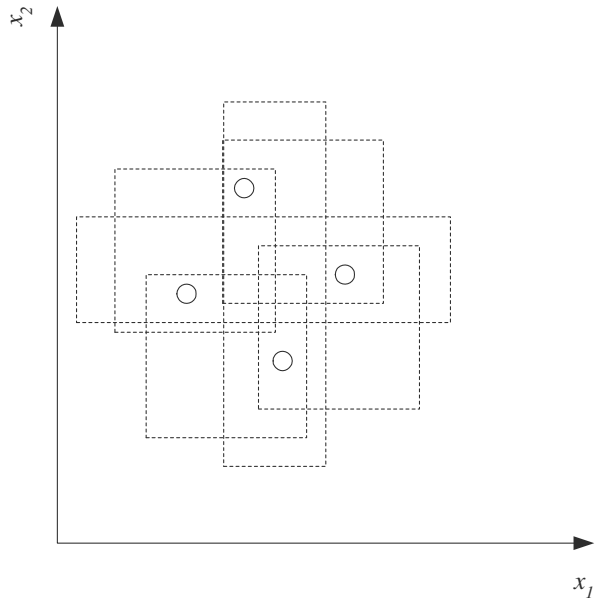

Figure 2.6 An axis-aligned rectangle can shatter four points. Only rectangles covering two points are shown.

mensions. For example, four points placed on a line cannot be shattered by rectangles. However, we cannot place five points in two dimensions *anywhere* such that a rectangle can separate the positive and negative examples for all possible labelings.

VC dimension may seem pessimistic. It tells us that using a rectangle as our hypothesis class, we can learn only datasets containing four points and not more. A learning algorithm that can learn datasets of four points is not very useful. However, this is because the VC dimension is independent of the probability distribution from which instances are drawn. In real life, the world is smoothly changing, instances close by most of the time have the same labels, and we need not worry about *all possible labelings*. There are a lot of datasets containing many more data points than four that are learnable by our hypothesis class (figure 2.1). So even hypothesis classes with small VC dimensions are applicable and are preferred over those with large VC dimensions, for example, a lookup table that has infinite VC dimension.

2.3 Probably Approximately Correct (PAC) Learning

Using the tightest rectangle, S, as our hypothesis, we would like to find how many examples we need. We would like our hypothesis to be approximately correct, namely, that the error probability be bounded by some value. We also would like to be confident in our hypothesis in that we want to know that our hypothesis will be correct most of the time (if not always); so we want to be probably correct as well (by a probability we can specify).

PAC LEARNING

In *Probably Approximately Correct* (PAC) *learning*, given a class, C, and examples drawn from some unknown but fixed probability distribution, $p(x)$, we want to find the number of examples, N, such that with probability at least $1-\delta$, the hypothesis h has error at most ϵ, for arbitrary $\delta \leq 1/2$ and $\epsilon > 0$

$$P\{C\Delta h \leq \epsilon\} \geq 1-\delta$$

where $C\Delta h$ is the region of difference between C and h.

In our case, because S is the tightest possible rectangle, the error region between C and $h = S$ is the sum of four rectangular strips (see figure 2.7). We would like to make sure that the probability of a positive example falling in here (and causing an error) is at most ϵ. For any of these strips, if we can guarantee that the probability is upper bounded by $\epsilon/4$, the error is at most $4(\epsilon/4) = \epsilon$. Note that we count the overlaps in the corners twice, and the total actual error in this case is less than $4(\epsilon/4)$. The probability that a randomly drawn example misses this strip is $1-\epsilon/4$. The probability that all N independent draws miss the strip is $(1-\epsilon/4)^N$, and the probability that all N independent draws miss any of the four strips is at most $4(1-\epsilon/4)^N$, which we would like to be at most δ. We have the inequality

$$(1-x) \leq \exp[-x]$$

So if we choose N and δ such that we have

$$4\exp[-\epsilon N/4] \leq \delta$$

we can also write $4(1-\epsilon/4)^N \leq \delta$. Dividing both sides by 4, taking (natural) log and rearranging terms, we have

$$N \geq (4/\epsilon)\log(4/\delta) \tag{2.7}$$

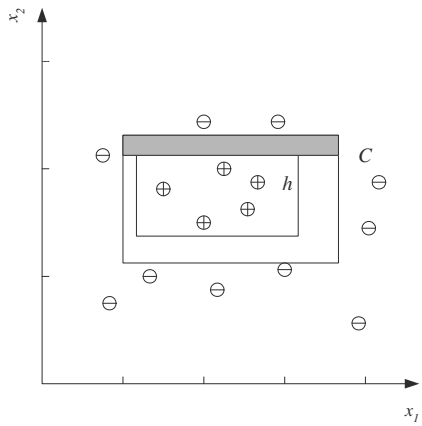

Figure 2.7 The difference between h and C is the sum of four rectangular strips, one of which is shaded.

Therefore, provided that we take at least $(4/\epsilon)\log(4/\delta)$ independent examples from C and use the tightest rectangle as our hypothesis h, with *confidence probability* at least $1-\delta$, a given point will be misclassified with *error probability* at most ϵ. We can have arbitrary large confidence by decreasing δ and arbitrary small error by decreasing ϵ, and we see in equation 2.7 that the number of examples is a slowly growing function of $1/\epsilon$ and $1/\delta$, linear and logarithmic, respectively.

2.4 Noise

NOISE

Noise is any unwanted anomaly in the data and due to noise, the class may be more difficult to learn and zero error may be infeasible with a simple hypothesis class (see figure 2.8). There are several interpretations of noise:

- There may be imprecision in recording the input attributes, which may shift the data points in the input space.
- There may be errors in labeling the data points, which may relabel

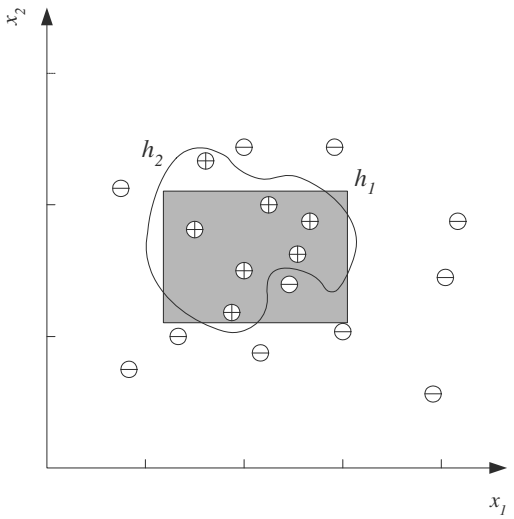

Figure 2.8 When there is noise, there is not a simple boundary between the positive and negative instances, and zero misclassification error may not be possible with a simple hypothesis. A rectangle is a simple hypothesis with four parameters defining the corners. An arbitrary closed form can be drawn by piecewise functions with a larger number of control points.

positive instances as negative and vice versa. This is sometimes called *teacher noise*.

- There may be additional attributes, which we have not taken into account, that affect the label of an instance. Such attributes may be *hidden* or *latent* in that they may be unobservable. The effect of these neglected attributes is thus modeled as a random component and is included in "noise."

As can be seen in figure 2.8, when there is noise, there is not a simple boundary between the positive and negative instances and to separate them, one needs a complicated hypothesis that corresponds to a hypothesis class with larger capacity. A rectangle can be defined by four numbers, but to define a more complicated shape one needs a more complex model with a much larger number of parameters. With a complex model,

one can make a perfect fit to the data and attain zero error; see the wiggly shape in figure 2.8. Another possibility is to keep the model simple and allow some error; see the rectangle in figure 2.8.

Using the simple rectangle (unless its training error is much bigger) makes more sense because of the following:

1. It is a simple model to use. It is easy to check whether a point is inside or outside a rectangle and we can easily check, for a future data instance, whether it is a positive or a negative instance.

2. It is a simple model to train and has fewer parameters. It is easier to find the corner values of a rectangle than the control points of an arbitrary shape. With a small training set when the training instances differ a little bit, we expect the simpler model to change less than a complex model: A simple model is thus said to have less *variance*. On the other hand, a too simple model assumes more, is more rigid, and may fail if indeed the underlying class is not that simple: A simpler model has more *bias*. Finding the optimal model corresponds to minimizing both the bias and the variance.

3. It is a simple model to explain. A rectangle simply corresponds to defining intervals on the two attributes. By learning a simple model, we can extract information from the raw data given in the training set.

4. If indeed there is mislabeling or noise in input and the actual class is really a simple model like the rectangle, then the simple rectangle, because it has less variance and is less affected by single instances, will be a better discriminator than the wiggly shape, although the simple one may make slightly more errors on the training set. Given comparable empirical error, we say that a simple (but not too simple) model would generalize better than a complex model. This principle is known as *Occam's razor*, which states that *simpler explanations are more plausible* and any unnecessary complexity should be shaved off.

OCCAM'S RAZOR

2.5 Learning Multiple Classes

In our example of learning a family car, we have positive examples belonging to the class family car and the negative examples belonging to all other cars. This is a *two-class* problem. In the general case, we have K

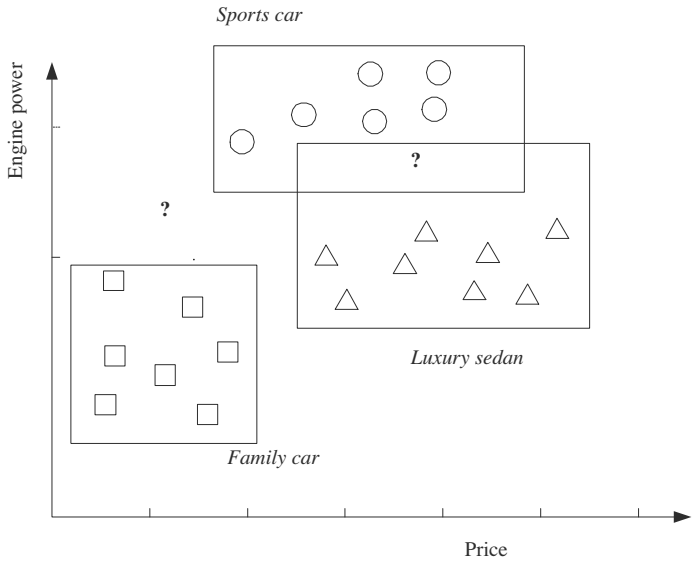

Figure 2.9 There are three classes: family car, sports car, and luxury sedan. There are three hypotheses induced, each one covering the instances of one class and leaving outside the instances of the other two classes. '?' are reject regions where no, or more than one, class is chosen.

classes denoted as $C_i, i = 1, \ldots, K$, and an input instance belongs to one and exactly one of them. The training set is now of the form

$$\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{r}^t\}_{t=1}^N$$

where $\boldsymbol{r}$ has K dimensions and

$$r_i^t = \begin{cases} 1 & \text{if } \boldsymbol{x}^t \in C_i \\ 0 & \text{if } \boldsymbol{x}^t \in C_j, j \neq i \end{cases} \tag{2.8}$$

An example is given in figure 2.9 with instances from three classes: family car, sports car, and luxury sedan.

In machine learning for classification, we would like to learn the boundary separating the instances of one class from the instances of all other classes. Thus we view a K-class classification problem as K two-class problems. The training examples belonging to C_i are the positive instances of hypothesis h_i and the examples of all other classes are the

negative instances of h_i. Thus in a K-class problem, we have K hypotheses to learn such that

$$h_i(\boldsymbol{x}^t) = \begin{cases} 1 & \text{if } \boldsymbol{x}^t \in C_i \\ 0 & \text{if } \boldsymbol{x}^t \in C_j, j \neq i \end{cases} \tag{2.9}$$

The total empirical error takes a sum over the predictions for all classes over all instances:

$$E(\{h_i\}_{i=1}^K | \mathcal{X}) = \sum_{t=1}^{N} \sum_{i=1}^{K} 1(h_i(\boldsymbol{x}^t) \neq r_i^t) \tag{2.10}$$

For a given $\boldsymbol{x}$, ideally only one of $h_i(\boldsymbol{x}), i = 1, \ldots, K$ is 1 and we can choose a class. But when no, or two or more, $h_i(\boldsymbol{x})$ is 1, we cannot choose a class, and this is the case of *doubt* and the classifier *rejects* such cases.

REJECT

In our example of learning a family car, we used only one hypothesis and only modeled the positive examples. Any negative example outside is not a family car. Alternatively, sometimes we may prefer to build two hypotheses, one for the positive and the other for the negative instances. This assumes a structure also for the negative instances that can be covered by another hypothesis. Separating family cars from sports cars is such a problem; each class has a structure of its own. The advantage is that if the input is a luxury sedan, we can have both hypotheses decide negative and reject the input.

If in a dataset, we expect to have all classes with similar distribution—shapes in the input space—then the same hypothesis class can be used for all classes. For example, in a handwritten digit recognition dataset, we would expect all digits to have similar distributions. But in a medical diagnosis dataset, for example, where we have two classes for sick and healthy people, we may have completely different distributions for the two classes; there may be multiple ways for a person to be sick, reflected differently in the inputs: All healthy people are alike; each sick person is sick in his or her own way.

2.6 Regression

In classification, given an input, the output that is generated is Boolean; it is a yes/no answer. When the output is a numeric value, what we would like to learn is not a class, $C(\boldsymbol{x}) \in \{0, 1\}$, but is a numeric function. In

machine learning, the function is not known but we have a training set of examples drawn from it

$$\mathcal{X} = \{\boldsymbol{x}^t, r^t\}_{t=1}^N$$

INTERPOLATION

where $r^t \in \Re$. If there is no noise, the task is *interpolation*. We would like to find the function $f(x)$ that passes through these points such that we have

$$r^t = f(\boldsymbol{x}^t)$$

In *polynomial interpolation*, given N points, we find the $(N-1)$st degree polynomial that we can use to predict the output for any $\boldsymbol{x}$. This is called

EXTRAPOLATION

extrapolation if $\boldsymbol{x}$ is outside of the range of $\boldsymbol{x}^t$ in the training set. In time-series prediction, for example, we have data up to the present and

REGRESSION

we want to predict the value for the future. In *regression*, there is noise added to the output of the unknown function

$$r^t = f(\boldsymbol{x}^t) + \epsilon \tag{2.11}$$

where $f(\boldsymbol{x}) \in \Re$ is the unknown function and ϵ is random noise. The explanation for noise is that there are extra *hidden* variables that we cannot observe

$$r^t = f^*(\boldsymbol{x}^t, \boldsymbol{z}^t) \tag{2.12}$$

where $\boldsymbol{z}^t$ denote those hidden variables. We would like to approximate the output by our model $g(\boldsymbol{x})$. The empirical error on the training set $\mathcal{X}$ is

$$E(g|\mathcal{X}) = \frac{1}{N}\sum_{t=1}^{N}[r^t - g(\boldsymbol{x}^t)]^2 \tag{2.13}$$

Because r and $g(\boldsymbol{x})$ are numeric quantities, for example, $\in \Re$, there is an ordering defined on their values and we can define a *distance* between values, as the square of the difference, which gives us more information than equal/not equal, as used in classification. The square of the difference is one error (loss) function that can be used; another is the absolute value of the difference. We will see other examples in the coming chapters.

Our aim is to find $g(\cdot)$ that minimizes the empirical error. Again our approach is the same; we assume a hypothesis class for $g(\cdot)$ with a small set of parameters. If we assume that $g(\boldsymbol{x})$ is linear, we have

$$g(\boldsymbol{x}) = w_1 x_1 + \cdots + w_d x_d + w_0 = \sum_{j=1}^{d} w_j x_j + w_0 \tag{2.14}$$

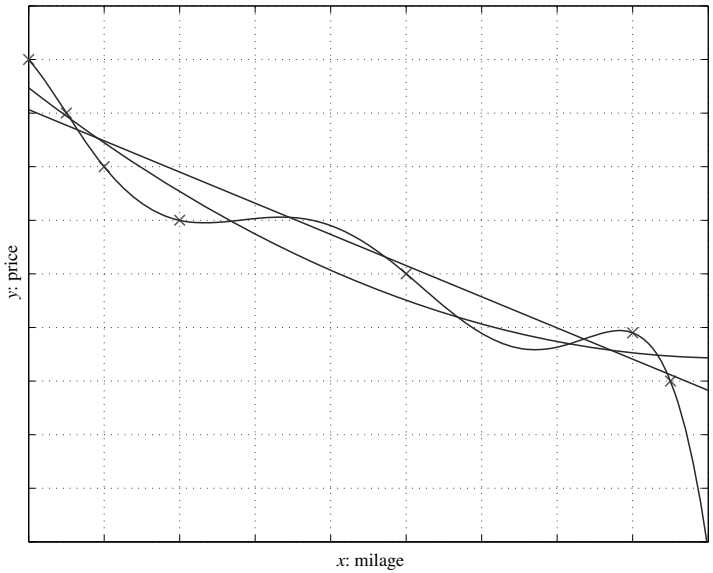

Figure 2.10 Linear, second-order, and sixth-order polynomials are fitted to the same set of points. The highest order gives a perfect fit, but given this much data it is very unlikely that the real curve is so shaped. The second order seems better than the linear fit in capturing the trend in the training data.

Let us now go back to our example in section 1.2.3 where we estimated the price of a used car. There we used a single input linear model

$$g(x) = w_1 x + w_0 \tag{2.15}$$

where w_1 and w_0 are the parameters to learn from data. The w_1 and w_0 values should minimize

$$E(w_1, w_0|\mathcal{X}) = \frac{1}{N}\sum_{t=1}^{N}[r^t - (w_1 x^t + w_0)]^2 \tag{2.16}$$

Its minimum point can be calculated by taking the partial derivatives of E with respect to w_1 and w_0, setting them equal to 0, and solving for the two unknowns:

$$\begin{aligned} w_1 &= \frac{\sum_t x^t r^t - \overline{xr}N}{\sum_t (x^t)^2 - N\overline{x}^2} \\ w_0 &= \overline{r} - w_1 \overline{x} \end{aligned} \tag{2.17}$$

Table 2.1 With two inputs, there are four possible cases and sixteen possible Boolean functions.

x_1	x_2	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}	h_{11}	h_{12}	h_{13}	h_{14}	h_{15}	h_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

where $\overline{x} = \sum_t x^t / N$ and $\overline{r} = \sum_t r^t / N$. The line found is shown in figure 1.2.

If the linear model is too simple, it is too constrained and incurs a large approximation error, and in such a case, the output may be taken as a higher-order function of the input—for example, quadratic

$$g(x) = w_2 x^2 + w_1 x + w_0 \tag{2.18}$$

where similarly we have an analytical solution for the parameters. When the order of the polynomial is increased, the error on the training data decreases. But a high-order polynomial follows individual examples closely, instead of capturing the general trend; see the sixth-order polynomial in figure 2.10. This implies that Occam's razor also applies in the case of regression and we should be careful when fine-tuning the model complexity to match it with the complexity of the function underlying the data.

2.7 Model Selection and Generalization

Let us start with the case of learning a Boolean function from examples. In a Boolean function, all inputs and the output are binary. There are 2^d possible ways to write d binary values and therefore, with d inputs, the training set has at most 2^d examples. As shown in table 2.1, each of these can be labeled as 0 or 1, and therefore, there are 2^{2^d} possible Boolean functions of d inputs.

Each distinct training example removes half the hypotheses, namely, those whose guesses are wrong. For example, let us say we have $x_1 = 0$, $x_2 = 1$ and the output is 0; this removes $h_5, h_6, h_7, h_8, h_{13}, h_{14}, h_{15}, h_{16}$. This is one way to interpret learning: we start with all possible hypothesis and as we see more training examples, we remove those hypotheses

that are not consistent with the training data. In the case of a Boolean function, to end up with a single hypothesis we need to see *all* 2^d training examples. If the training set we are given contains only a small subset of all possible instances, as it generally does—that is, if we know what the output should be for only a small percentage of the cases—the solution is not unique. After seeing N example cases, there remain 2^{2^d-N} possible functions. This is an example of an *ill-posed problem* where the data by itself is not sufficient to find a unique solution.

ILL-POSED PROBLEM

The same problem also exists in other learning applications, in classification, and in regression. As we see more training examples, we know more about the underlying function, and we carve out more hypotheses that are inconsistent from the hypothesis class, but we still are left with many consistent hypotheses.

So because learning is ill-posed, and data by itself is not sufficient to find the solution, we should make some extra assumptions to have a unique solution with the data we have. The set of assumptions we make to have learning possible is called the *inductive bias* of the learning algorithm. One way we introduce inductive bias is when we assume a hypothesis class $\mathcal{H}$. In learning the class of family car, there are infinitely many ways of separating the positive examples from the negative examples. Assuming the shape of a rectangle is one inductive bias, and then the rectangle with the largest margin for example, is another inductive bias. In linear regression, assuming a linear function is an inductive bias, and among all lines, choosing the one that minimizes squared error is another inductive bias.

INDUCTIVE BIAS

But we know that each hypothesis class has a certain capacity and can learn only certain functions. The class of functions that can be learned can be extended by using a hypothesis class with larger capacity, containing more complex hypotheses. For example, the hypothesis class that is a union of two rectangles has higher capacity, but its hypotheses are more complex. Similarly in regression, as we increase the order of the polynomial, the capacity and complexity increase. The question now is to decide where to stop.

Thus learning is not possible without inductive bias, and now the question is how to choose the right bias. This is called *model selection*, which is choosing between possible $\mathcal{H}$. In answering this question, we should remember that the aim of machine learning is rarely to replicate the training data but the prediction for new cases. That is we would like to be able to generate the right output for an input instance outside the training set,

MODEL SELECTION

one for which the correct output is not given in the training set. How well a model trained on the training set predicts the right output for new instances is called *generalization*.

GENERALIZATION

For best generalization, we should match the complexity of the hypothesis class $\mathcal{H}$ with the complexity of the function underlying the data. If $\mathcal{H}$ is less complex than the function, we have *underfitting*, for example, when trying to fit a line to data sampled from a third-order polynomial. In such a case, as we increase the complexity, the training error decreases. But if we have $\mathcal{H}$ that is too complex, the data is not enough to constrain it and we may end up with a bad hypothesis, $h \in \mathcal{H}$, for example, when fitting two rectangles to data sampled from one rectangle. Or if there is noise, an overcomplex hypothesis may learn not only the underlying function but also the noise in the data and may make a bad fit, for example, when fitting a sixth-order polynomial to noisy data sampled from a third-order polynomial. This is called *overfitting*. In such a case, having more training data helps but only up to a certain point. Given a training set and $\mathcal{H}$, we can find $h \in \mathcal{H}$ that has the minimum training error but if $\mathcal{H}$ is not chosen well, no matter which $h \in \mathcal{H}$ we pick, we will not have good generalization.

UNDERFITTING

OVERFITTING

TRIPLE TRADE-OFF

We can summarize our discussion citing the *triple trade-off* (Dietterich 2003). In all learning algorithms that are trained from example data, there is a trade-off between three factors:

- the complexity of the hypothesis we fit to data, namely, the capacity of the hypothesis class,
- the amount of training data, and
- the generalization error on new examples.

As the amount of training data increases, the generalization error decreases. As the complexity of the model class $\mathcal{H}$ increases, the generalization error decreases first and then starts to increase. The generalization error of an overcomplex $\mathcal{H}$ can be kept in check by increasing the amount of training data but only up to a point. If the data is sampled from a line and if we are fitting a higher-order polynomial, the fit will be constrained to lie close to the line if there is training data in the vicinity; where it has not been trained, a high-order polynomial may behave erratically.

We can measure the generalization ability of a hypothesis, namely, the quality of its inductive bias, if we have access to data outside the training

set. We simulate this by dividing the training set we have into two parts. We use one part for training (i.e., to fit a hypothesis), and the remaining part is called the *validation set* and is used to test the generalization ability. That is, given a set of possible hypothesis classes $\mathcal{H}_i$, for each we fit the best $h_i \in \mathcal{H}_i$ on the training set. Then, assuming large enough training and validation sets, the hypothesis that is the most accurate on the validation set is the best one (the one that has the best inductive bias). This process is called *cross-validation*. So, for example, to find the right order in polynomial regression, given a number of candidate polynomials of different orders where polynomials of different orders correspond to $\mathcal{H}_i$, for each order, we find the coefficients on the training set, calculate their errors on the validation set, and take the one that has the least validation error as the best polynomial.

VALIDATION SET

CROSS-VALIDATION

Note that if we need to report the error to give an idea about the expected error of our best model, we should not use the validation error. We have used the validation set to choose the best model, and it has effectively become a part of the training set. We need a third set, a *test set*, sometimes also called the *publication set*, containing examples not used in training or validation. An analogy from our lives is when we are taking a course: the example problems that the instructor solves in class while teaching a subject form the training set; exam questions are the validation set; and the problems we solve in our later, professional life are the test set.

TEST SET

We cannot keep on using the same training/validation split either, because after having been used once, the validation set effectively becomes part of training data. This will be like an instructor who uses the same exam questions every year; a smart student will figure out not to bother with the lectures and will only memorize the answers to those questions.

We should always remember that the training data we use is a random sample, that is, for the same application, if we collect data once more, we will get a slightly different dataset, the fitted h will be slightly different and will have a slightly different validation error. Or if we have a fixed set which we divide for training, validation, and test, we will have different errors depending on how we do the division. These slight differences in error will allow us to estimate how large differences should be to be considered *significant* and not due to chance. That is, in choosing between two hypothesis classes $\mathcal{H}_i$ and $\mathcal{H}_j$, we will use them both multiple times on a number of training and validation sets and check if the difference between average errors of h_i and h_j is larger than the average difference

between multiple h_i. In chapter 19, we discuss how to design machine learning experiments using limited data to best answer our questions—for example, which is the best hypothesis class?—and how to analyze the results of these experiments so that we can achieve statistically significant conclusions minimally affected by random chance.

2.8 Dimensions of a Supervised Machine Learning Algorithm

Let us now recapitulate and generalize. We have a sample

$$\mathcal{X} = \{x^t, r^t\}_{t=1}^N \tag{2.19}$$

IID The sample is *independent and identically distributed* (*iid*); the ordering is not important and all instances are drawn from the same joint distribution $p(x, r)$. t indexes one of the N instances, x^t is the arbitrary dimensional input, and r^t is the associated desired output. r^t is 0/1 for two-class learning, is a K-dimensional binary vector (where exactly one of the dimensions is 1 and all others 0) for $(K > 2)$-class classification, and is a real value in regression.

The aim is to build a good and useful approximation to r^t using the model $g(x^t|\theta)$. In doing this, there are three decisions we must make:

1. *Model* we use in learning, denoted as

   $$g(x|\theta)$$

   where $g(\cdot)$ is the model, x is the input, and θ are the parameters.

   $g(\cdot)$ defines the hypothesis class $\mathcal{H}$, and a particular value of θ instantiates one hypothesis $h \in \mathcal{H}$. For example, in class learning, we have taken a rectangle as our model whose four coordinates make up θ; in linear regression, the model is the linear function of the input whose slope and intercept are the parameters learned from the data. The model (inductive bias), or $\mathcal{H}$, is fixed by the machine learning system designer based on his or her knowledge of the application and the hypothesis h ic chosen (parameters are tuned) by a learning algorithm using the training set, sampled from $p(x, r)$.

2. *Loss function*, $L(\cdot)$, to compute the difference between the desired output, r^t, and our approximation to it, $g(x^t|\theta)$, given the current value

of the parameters, θ. The *approximation error*, or *loss*, is the sum of losses over the individual instances

$$E(\theta|\mathcal{X}) = \sum_t L(r^t, g(x^t|\theta)) \tag{2.20}$$

In class learning where outputs are 0/1, $L(\cdot)$ checks for equality or not; in regression, because the output is a numeric value, we have ordering information for distance and one possibility is to use the square of the difference.

3. *Optimization procedure* to find θ^* that minimizes the total error

$$\theta^* = \arg\min_\theta E(\theta|\mathcal{X}) \tag{2.21}$$

where arg min returns the argument that minimizes. In regression, we can solve analytically for the optimum. With more complex models and error functions, we may need to use more complex optimization methods, for example, gradient-based methods, simulated annealing, or genetic algorithms.

For this to work well, the following conditions should be satisfied: First, the hypothesis class of $g(\cdot)$ should be large enough, that is, have enough capacity, to include the unknown function that generated the data that is represented in r^t in a noisy form. Second, there should be enough training data to allow us to pinpoint the correct (or a good enough) hypothesis from the hypothesis class. Third, we should have a good optimization method that finds the correct hypothesis given the training data.

Different machine learning algorithms differ either in the models they assume (their hypothesis class/inductive bias), the loss measures they employ, or the optimization procedure they use. We will see many examples in the coming chapters.

2.9 Notes

Mitchell proposed version spaces and the candidate elimination algorithm to incrementally build S and G as instances are given one by one; see Mitchell 1997 for a recent review. The rectangle-learning is from exercise 2.4 of Mitchell 1997. Hirsh (1990) discusses how version spaces can handle the case when instances are perturbed by small amount of noise.

In one of the earliest works on machine learning, Winston (1975) proposed the idea of a "near miss." A near miss is a negative example that is very much like a positive example. In our terminology, we see that a near miss would be an instance that falls in the gray area between S and G, an instance which would affect the margin, and would hence be more useful for learning, than an ordinary positive or negative example. The instances that are close to the boundary are the ones that define it (or support it); those which are surrounded by many instances with the same label can be added/removed without affecting the boundary.

Related to this idea is *active learning* where the learning algorithm can generate instances itself and ask for them to be labeled, instead of passively being given them (Angluin 1988) (see exercise 4).

VC dimension was proposed by Vapnik and Chervonenkis in the early 1970s. A recent source is Vapnik 1995 where he writes, "Nothing is more practical than a good theory" (p. x), which is as true in machine learning as in any other branch of science. You should not rush to the computer; you can save yourself from hours of useless programming by some thinking, a notebook, and a pencil—you may also need an eraser.

The PAC model was proposed by Valiant (1984). The PAC analysis of learning a rectangle is from Blumer et al. 1989. A good textbook on computational learning theory covering PAC learning and VC dimension is Kearns and Vazirani 1994.

2.10 Exercises

1. Let us say our hypothesis class is a circle instead of a rectangle. What are the parameters? How can the parameters of a circle hypothesis be calculated in such a case? What if it is an ellipse? Why does it make more sense to use an ellipse instead of a circle? How can you generalize your code to $K > 2$ classes?
2. Imagine our hypothesis is not one rectangle but a union of two (or $m > 1$) rectangles. What is the advantage of such a hypothesis class? Show that any class can be represented by such a hypothesis class with large enough m.
3. The complexity of most learning algorithms is a function of the training set. Can you propose a filtering algorithm that finds redundant instances?
4. If we have a supervisor who can provide us with the label for any $\boldsymbol{x}$, where should we choose $\boldsymbol{x}$ to learn with fewer queries?
5. In equation 2.13, we summed up the squares of the differences between the actual value and the estimated value. This error function is the one most

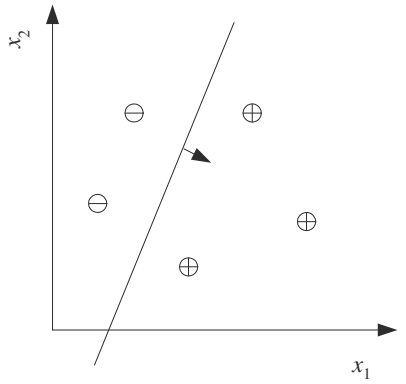

Figure 2.11 A line separating positive and negative instances.

frequently used, but it is one of several possible error functions. Because it sums up the squares of the differences, it is not robust to outliers. What would be a better error function to implement *robust regression*?

6. Derive equation 2.17.
7. Assume our hypothesis class is the set of lines, and we use a line to separate the positive and negative examples, instead of bounding the positive examples as in a rectangle, leaving the negatives outside (see figure 2.11). Show that the VC dimension of a line is 3.
8. Show that the VC dimension of the triangle hypothesis class is 7 in two dimensions. (Hint: For best separation, it is best to place the seven points equidistant on a circle.)
9. Assume as in exercise 7 that our hypothesis class is the set of lines. Write down an error function that not only minimizes the number of misclassifications but also maximizes the margin.
10. One source of noise is error in the labels. Can you propose a method to find data points that are highly likely to be mislabeled?

2.11 References

Angluin, D. 1988. "Queries and Concept Learning." *Machine Learning* 2: 319–342.

Blumer, A., A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. 1989. "Learnability and the Vapnik-Chervonenkis Dimension." *Journal of the ACM* 36: 929–965.

Dietterich, T. G. 2003. "Machine Learning." In *Nature Encyclopedia of Cognitive Science*. London: Macmillan.

Hirsh, H. 1990. *Incremental Version Space Merging: A General Framework for Concept Learning*. Boston: Kluwer.

Kearns, M. J., and U. V. Vazirani. 1994. *An Introduction to Computational Learning Theory*. Cambridge, MA: MIT Press.

Mitchell, T. 1997. *Machine Learning*. New York: McGraw-Hill.

Valiant, L. 1984. "A Theory of the Learnable." *Communications of the ACM* 27: 1134–1142.

Vapnik, V. N. 1995. *The Nature of Statistical Learning Theory*. New York: Springer.

Winston, P. H. 1975. "Learning Structural Descriptions from Examples." In *The Psychology of Computer Vision*, ed. P. H. Winston, 157–209. New York: McGraw-Hill.